

**EV355226673**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**INCREMENTAL NON-CHRONOLOGICAL  
SYNCHRONIZATION OF NAMESPACES**

Inventor(s):

John H. Zybura

Max L. Benson

Herman Man

Edward H. Wayt

Felix W. Wong

Jing Wu

ATTORNEY DOCKET NO. MS1-1688US

## **TECHNICAL FIELD**

This application relates generally to synchronization of information and more specifically to synchronization of information in a plurality of information structures or hierarchies.

## **BACKGROUND OF THE INVENTION**

Often a company stores important information in various data sources. For example, a human resources department may store information about employees in a human resources data source. The human resources data source may be arranged or organized according to a human resources specific information structure or hierarchy. A finance department may also store information about employees, clients, suppliers, etc., in a finance department data source. The finance department data source may be arranged or organized according to a finance department information structure or hierarchy. It is likely that some common information exists in both data sources. Thus, synchronizing the information becomes desirable.

A synchronizing process typically implements rules and/or specifications to adequately harmonize information in various data sources. Further, such a process may rely on an engine capable of executing software and a storage capable of storing the information, as appropriate. In general, the synchronizing process may replicate information from various data sources in a central storage, wherein the replicated information has some degree of integrity. To achieve this task, information from the various data sources are either pushed or pulled into the

1 central storage. In addition, information may be pulled or pushed out of such a  
2 central storage to the various data sources.

3  
4 Often, the information may be provided to the central storage by one of the  
5 various data sources non-chronologically. In other words, the modifications that  
6 occur at a data source have a temporal relationship. However, due to the nature of  
7 synchronization, notification of those modifications may reach the central storage  
8 out of order with respect to that temporal relationship. This situation has the  
9 potential to create problems during synchronization. Various exemplary methods,  
10 devices and/or systems described below are directed at those problems.

## 11 12 **SUMMARY OF THE INVENTION**

13 Briefly stated, mechanisms and techniques are described for enabling  
14 incremental non-chronological synchronization of namespaces. In an  
15 environment, entities must have unique names within a namespace and entities  
16 may only refer to entities that actually exist within the namespace. Synchronizing  
17 two such namespaces involves providing a mechanism for indicating that an entity  
18 has been created because a reference to that entity has been made even though that  
19 entity does not yet exist. At such time as the entity is formally created, the  
20 indication is removed. Synchronizing two such namespaces also involves  
21 providing a mechanism for indicating that an entity's unique name in the  
22 namespace has been compromised through the synchronization process.

## 23 24 **BRIEF DESCRIPTION OF THE DRAWINGS**

25

1 Fig. 1 is a functional block diagram generally illustrating an exemplary  
2 system that includes a metadirectory and a plurality of data sources.

3 Fig. 2 is a functional block diagram illustrating in slightly greater detail the  
4 storage of the metadirectory as it interacts with various data sources.

5 Fig. 3 is a functional block diagram generally illustrating information that is  
6 included in an "entity" as that term is used in this document.

7 Fig. 4 is a graphical representation of a mechanism for addressing name  
8 collision during an incremental non-chronological synchronization.

9 Fig. 5 is a graphical illustration of a synchronization between a master  
10 namespace and a slave namespace that suffers from name collision.

11 Fig. 6 is a graphical illustration of another synchronization between a  
12 master namespace and a slave namespace that also suffers from name collision.

13 Fig. 7 is a graphical illustration of a synchronization between a master  
14 namespace and a slave namespace that suffers from a dangling reference.

15 Fig. 8 shows an exemplary computer suitable as an environment for  
16 practicing various aspects of subject matter disclosed herein.

## 17 18 **DETAILED DESCRIPTION**

19 The following description sets forth a specific embodiment of a system for  
20 incremental non-chronological synchronization. This specific embodiment  
21 incorporates elements recited in the appended claims. The embodiment is  
22 described with specificity in order to meet statutory requirements. However, the  
23 description itself is not intended to limit the scope of this patent. Rather, the  
24 inventors have contemplated that the claimed invention might also be embodied in  
25 other ways, to include different elements or combinations of elements similar

1 to the ones described in this document, in conjunction with other present or future  
2 technologies.

3  
4 The following discussion refers to an information environment that  
5 includes a metadirectory. While a metadirectory is used here for explanatory  
6 purposes, the various mechanisms and techniques described here may also be  
7 applied generally to other environments where synchronization of information is  
8 desired. In general, information should be identifiable in an information  
9 environment, for example, through use of an identifier, and preferably an  
10 immutable or traceable identifier. In some instances, information is structured or  
11 organized in a hierarchy.

#### 12 13 Exemplary Metadirectory System

14 Fig. 1 shows an exemplary system 100 that includes an exemplary  
15 metadirectory 102 capable of communicating information to and/or from a  
16 plurality of data sources (e.g., DSA 150, DSB 160, and DSC 170). Each data  
17 source includes many objects, with each object containing information. For this  
18 discussion, each object may be thought of as a body of information, such as  
19 information about an individual (e.g., name, address, salary), a mailing list  
20 (members), an e-mail account (e-mail address), a corporate asset (serial number),  
21 or the like. For example if DSA 150 were a human resources database, then  
22 objects within DSA 150 may correspond to employees, and each employee may  
23 have characteristics such as an employee number, a manager, an office location,  
24 and the like.  
25

1        There may also be an object in another data source that pertains to the same  
2 body of information, but includes slightly different characteristics or information.  
3 For example, DSB 160 may be an information technology server that includes  
4 information about the logon accounts of employees. Accordingly, there may be a  
5 corresponding object within DSB 160 for each or many of the objects in DSA 150.  
6 However, the particular body of information for the objects within DSB 160 would  
7 be slightly different than those within DSA 150. Collectively, the information  
8 associated with a particular body of information are sometimes referred to as  
9 “identity data” or the like.

10  
11        The metadirectory 102 is an infrastructure element that provides an  
12 aggregation and clearinghouse of the information stored within each of the several  
13 data sources associated with the metadirectory 102. The metadirectory 102  
14 includes storage 130 in which reside “entities” that represent the individual bodies  
15 of information stored in each associated data source. Disparate information from  
16 different data sources that pertains to the same body of information (e.g., an  
17 individual, asset, or the like) is typically aggregated into a single entity within the  
18 metadirectory 102. In this way, a user can take advantage of the  
19 metadirectory 102 to view at a single location information that may be stored  
20 piecemeal in several different data sources. Such an exemplary metadirectory may  
21 consolidate information contained in multiple data sources in a centralized  
22 manner, manage relationships between the data sources, and allow for information  
23 to flow between them as appropriate.

1       The metadirectory 102 includes rules 110 and services 120 that are used to  
2 aggregate, consolidate, synchronize, and otherwise maintain the integrity of the  
3 information presented through the metadirectory 102. The rules 110 and  
4 services 120 form or define one or more protocols, APIs, schemata, services,  
5 hierarchies, etc. In this particular embodiment, the rules 110 include methods and  
6 techniques for achieving incremental non-chronological synchronization of  
7 information presented to the metadirectory 102, as will become more clear in the  
8 description that follows.

9  
10       The storage 130 is for storing the aggregated and consolidated information  
11 from each of the associated data sources. The storage 130 may be a database or  
12 any other mechanism for persisting data in a substantially permanent manner. As  
13 will be described more fully in conjunction with Fig. 2, the storage 130 may  
14 include core storage (sometimes referred to as a “metaverse”), in which the data is  
15 deemed to be valid, and transient storage (e.g., a buffer or connector space) used to  
16 temporarily store information awaiting inclusion in the core storage. In other  
17 words, changes, additions, or deletions to information in one or more data sources  
18 may be presented to the metadirectory 102 and temporarily stored in a buffer until  
19 they can be committed to the core storage.

20  
21       Fig. 2 is a functional block diagram illustrating in slightly greater detail the  
22 storage 130 of the metadirectory 102 as it interacts with the various data sources.  
23 The data stored within the system are termed “entities” for the purpose of this  
24 discussion (e.g., core entity 270, buffer entity 260, external entity 250). Generally  
25 stated, entities are objects that include any arbitrary collection of data (e.g., current

1 values, change information, etc.) about the bodies of information that reside in the  
2 various data sources. Entities within the metadirectory 102 may be referred to  
3 collectively as "central entities," and entities outside the metadirectory 102 (e.g.,  
4 within the data sources) may be referred to collectively as "external entities." For  
5 example, a central entity within the metadirectory 102 may correspond to two or  
6 more external entities and include an aggregation of the information stored in each  
7 of the corresponding external entities. More specific detail about entities is  
8 provided below in conjunction with Fig. 3.

9  
10 As mentioned above, the storage 130 may include a core 211 and a buffer  
11 221. The core 211 represents data that is considered to accurately reflect (from the  
12 perspective of a user of the metadirectory 102) the information in the various data  
13 sources. In contrast, the buffer 221 includes data of a more transient nature.  
14 Recent changes to data at the data sources are reflected in the buffer 221 until that  
15 data can be committed to the core 211.

16  
17 As illustrated, a data source (e.g., DSA 150) presents to the  
18 metadirectory 102 change data that represents changes to an external entity (e.g.,  
19 external entity 250) stored within the data source. The change data may indicate  
20 that information about an entity within the data source has changed in some  
21 fashion, or perhaps the change data indicates that its corresponding entity has been  
22 deleted or added. A buffer entity (e.g., buffer entity 260) is first created using the  
23 change data to create an entity that represents the external entity (e.g., external  
24 entity 250) at the data source (e.g., DSA 150). Essentially, the buffer entity 260  
25 mirrors its corresponding external entity 250. Other data sources (not shown) may



1 also be presenting their own change data to the buffer 221 as well. The change  
2 data may sometimes be referred to as "delta information" or "deltas."

3  
4 As used in this document, the term "namespace" means any set of entities.  
5 The entities in a namespace may be unordered. Accordingly the term namespace  
6 may be used to refer to any set of entities, such as the core 211 or the buffer 221  
7 (i.e., a core namespace or a buffer namespace). Or the term namespace may be  
8 used to refer collectively to the metadirectory 102 as a namespace. Similarly, any  
9 of the data sources may sometimes be referred to as namespaces.

10  
11 A process termed synchronization occurs to reconcile the external entities  
12 within the data sources with their corresponding central entities within the  
13 metadirectory 102. For instance, in this example the external entity 250 is  
14 associated with the buffer entity 260. Through the synchronization process,  
15 modifications represented in the external entity 250, as well as perhaps other  
16 external entities, become reflected in the buffer entity 260. By creating this  
17 synchronized relationship between two namespaces (e.g., the buffer 221 and the  
18 DSA 150), the pair of namespaces may be referred to as "correlated namespaces."

19  
20 Synchronization may be more generally defined as any process which  
21 causes two non-identical namespaces (e.g., the buffer 221 and the DSA 150) to  
22 converge to identity over a finite period of time. In various examples in this  
23 document, one namespace is sometimes referred to as the master, and it is allowed  
24 to be modified by other processes. Using the terminology of this discussion, the  
25 DSA 150 is the master. In this example, the other namespace is referred to as the

1 slave, and it may only be modified by the synchronization process. Again, using  
2 the terminology of this discussion, the buffer 221 is the slave.

3  
4 Typically, the synchronization process is not instantaneous. That is, the  
5 time to convergence for the system is nonzero. Such synchronization processes  
6 may be termed periodic, since the non-zero convergence time generally means the  
7 synchronization processes itself is not continuously active, but rather is invoked on  
8 a schedule. Thus, a periodic synchronization process could be described by the  
9 following steps:

10  
11 1) Select from the master (e.g., the DSA 150) a set of changes to apply to  
12 the slave (e.g., the buffer 221) that is sufficient to bring the two  
13 namespaces into convergence. Thus the set should include all changes  
14 made to the master since the last time it synchronized with the slave. In  
15 simple implementations, this set could simply be the set of all entities in  
16 the master, where each entity is represented as a modify-entity event  
17 which would set all properties. The resultant set is called the  
18 synchronization feed.

19  
20 2) Iterate, in arbitrary order, through the set of changes (i.e., consume the  
21 feed) obtained in the previous step. If the change is add-entity, then an  
22 entity is created in the slave namespace. Otherwise, apply the change to  
23 the appropriate entity in the slave namespace.

1 Finally, a periodic synchronization process is said to be incremental if each  
2 change read from the synchronization feed is performed independently on the  
3 slave namespace. By contrast, a non-incremental synchronization process would  
4 be one in which the entire summation of all changes in the feed are performed in a  
5 single, atomic update. In practice, virtually all synchronization processes are  
6 incremental. The amount of data which may need to be transferred during a given  
7 invocation of the synchronization process is effectively unbounded, which would  
8 make it prohibitively expensive to use batching or transactional logic in the  
9 underlying data source to achieve an atomic write from the summation of all  
10 synchronization modifications.

11  
12 The general process of incremental synchronization as just described allows  
13 for the possibility that the changes in the synchronization feed could be applied in  
14 non-chronological order. That is to say, the order of changes applied to the slave  
15 as a result of processing the synchronization feed is not necessarily the same as the  
16 order in which the changes were originally applied to the master namespace. In  
17 fact, for the simplest implementation where simply all entities in the master  
18 namespace are selected every time, the resultant will feed will almost always be  
19 non-chronological.

20  
21 Consuming a non-chronological feed can result in temporary data artifacts  
22 in the slave namespace which could violate constraints placed on a namespace.  
23 These artifacts are described as temporary because they can only be seen if the  
24 state of the slave namespace is viewed after each individual change in the feed is  
25 consumed; the artifacts should not remain once the synchronization feed has been

1 depleted. Thus, the artifacts are not problematic if the synchronization process is  
2 non-incremental, since the final state of the namespace written to the data store  
3 would be consistent. However, these artifacts do pose problems for incremental  
4 synchronization processes since each individual change is performed in the slave  
5 namespace independently, and any of these changes could violate the namespace  
6 constraints and leave the slave in an inconsistent state.

7  
8 Specifically, two types of data artifacts can occur when incrementally  
9 consuming a non-chronological synchronization feed, name collision and dangling  
10 references. Name collision occurs when an external entity is processed which  
11 either adds a new buffer entity or changes the name of an existing buffer entity  
12 and results in duplicate names, which violates the namespace constraint that all  
13 names be unique at a single point in time. Examples of how name collision can  
14 occur are illustrated in Figs. 5 and 6 and described below. Dangling references  
15 occur when a buffer entity refers to another entity that does not yet exist because  
16 the change that would create it has not yet been processed. A dangling reference  
17 violates the namespace constraint that all reference values contain the names of  
18 existing entities. An example of how a dangling reference can occur is illustrated  
19 in Fig. 7 and described below.

20  
21 Note that these constraint violations are only temporary artifacts of the  
22 synchronization process. A successfully completed synchronization process  
23 should not exhibit these problems. However, for an incremental synchronization  
24 process, these temporary artifacts can adversely affect the system during  
25 intermediate stages between separately transacted modifications made to the slave

1 namespace. Most data sources would not allow the temporary relaxation of these  
2 types of constraints. And the fact that the incremental synchronization process is  
3 non-atomic means an aborted or unsuccessfully completed process could result in  
4 these artifacts persisting after the process has ended.

5  
6 Fig. 3 is a functional block diagram generally illustrating information that is  
7 included in an "entity" 310 as that term is used in this document. The entity 310  
8 includes a name (e.g., name 311), which preferably has a string value 321 unique  
9 across the namespace. The name can change at any time. Each entity also  
10 includes an identity (e.g., identity 312), which is preferably a string value 322 that  
11 is globally unique. The identity of an entity does not change, i.e. it is an  
12 immutable property of the entity 310.

13  
14 The use of both a name and a unique identifier may at first appear  
15 redundant, but each has a special purpose. For example, a human-readable name  
16 is intuitive and may be used to reflect a real-world property or concept, thus  
17 making the name very useful to users. However, this usability typically means  
18 that the name should also be changeable. In contrast, a globally unique identifier  
19 conveys little in terms of readability or intuitive message. It does however,  
20 effectively distinguish the entity from every other entity in existence.

21  
22 The entity 310 may also include an arbitrary number of reference attributes  
23 (e.g., reference attribute 313) that contain name/identity pairs 323 of other entities  
24 within the same namespace referred to by the referring entity. The reference  
25 attribute 313 may have a single reference pair, or it may include multiple reference

1 pairs, such as a distribution list. The reference attributes allow the modeling of  
2 arbitrary, directed relationships between entities. The entity 310 may also include  
3 an arbitrary number of user data attributes (e.g., data\_1 314 and data\_2 315) that  
4 contain user data (e.g., user info 324 and 325, respectively).

5  
6 The entity 310 also includes a “phantom” attribute 316, which has special  
7 meaning in the context of this discussion. As described above, the process of  
8 incremental non-chronological synchronization can result in dangling references  
9 (an entity that does not yet exist is referred to by a changed entity). The phantom  
10 attribute 316 is a Boolean-valued property 326 of the entity 310 used to indicate  
11 that the entity has not yet been officially “created” but yet must exist in the  
12 namespace because it has been referred to by another entity. Use of the phantom  
13 attribute allows the creation of a “placeholder state,” which is essentially  
14 somewhere between an officially-created entity and a non-existent entity. Other  
15 constraints may be put on the entity 310 if the phantom property 316 is true. For  
16 instance, no other data may be allowed to be stored in a phantom entity except the  
17 name 311 and identity 312. Use of the placeholder state is illustrated in Fig. 7.

18  
19 The following guidance is provided for handling phantoms or entities in the  
20 placeholder state. First, if an add-entity event occurs and there already exists a  
21 phantom entity with the same identity, then the phantom entity is promoted to a  
22 non-placeholder entity. If a delete-entity event occurs and the entity to be deleted  
23 has references to it in the slave namespace, then the entity is demoted to a  
24 phantom entity.

1        If a change to an entity occurs that adds a value to a reference attribute and  
2 the referent does not exist, then a phantom entity is created with the correct name  
3 and identity and is used as the referent. If a change to an entity occurs that deletes  
4 a value from a reference attribute and a phantom entity was being referred to, the  
5 phantom entity is checked to see if it still has anything else referring to it. If not,  
6 then the phantom entity is deleted. Note that this process does not have to be  
7 immediate. Rather, at the end of the synchronization process, a final sweep can be  
8 made to find any phantom entities without references to them. If found, these  
9 orphaned phantom entities may then be deleted.

10  
11        Fig. 4 is a graphical representation of a mechanism for addressing name  
12 collision during an incremental non-chronological synchronization. More  
13 specifically, illustrated is a slave namespace partitioned into two subspaces, a  
14 transient subspace 410 and a non-transient subspace 411. The non-transient  
15 subspace 411 is configured sufficient to contain every possible name in the master  
16 namespace. Accordingly, the non-transient subspace 411 is sufficient to include  
17 every entity (e.g., entity 450) in the slave namespace if every single entity is  
18 uniquely named (as is the constraint) and is non-transient.

19  
20        Recall that name collision occurs when an entity (which could be a  
21 phantom entity) is introduced that violates the constraint that no two entities  
22 within a namespace have the same name. Name collision can occur if the name of  
23 an existing entity is changed to an already taken name or if a new entity (which  
24 could be a phantom entity) is given a name that conflicts with an existing name  
25 (possibly the name of an existing phantom entity) prior to another change that

1 would resolve the conflict. To address this situation, the entity is moved to the  
2 transient namespace 410 by changing the name of the existing entity to some value  
3 that cannot conflict with other entities. The use of the entity's identity (which is  
4 globally unique) in combination with (or in lieu of) the entity's name would  
5 suffice to prevent name collision. Accordingly, the transient subspace 410 is  
6 configured sufficient to contain every possible identity value in the master  
7 namespace. Upon being identified as transient, an entity's name may be changed  
8 to its identity or a combination of its identity and its name.

9  
10 Another Boolean-valued flag in the data store for the slave namespace may  
11 be used to indicate that an entity is transient. The flag in combination with the  
12 string-valued name may also be used as a logical unique "name" for transient  
13 entities. For example, entity 451 has a flag 461 that is currently set False and a  
14 name "Foo." Upon being identified as transient, the entity 451' is moved to the  
15 transient subspace 410 by setting the flag 461' to True. In addition, the name of  
16 the entity is changed to its identifier "GUID."

17  
18 The following guidance is provided for handling transient entities, or  
19 entities that result from a name collision. If an add-entity event occurs and there  
20 already exists an entity with the same name but a different identity, the existing  
21 entity is made transient. If a entity's name is changed but there already exists an  
22 entity with the new name but a different identity, the existing entity is made  
23 transient. A modify-entity event should be treated as a name change if the entity to  
24 which the event refers is currently transient. That is, receiving a change to an  
25 entity currently identified as transient should bring it out of the transient state.



1 Note that bringing an entity out of the transient state should be treated the same as  
2 any other change. Any existing entities with conflicting names should in turn be  
3 made transient.

4  
5 The concepts and principles of these mechanisms and techniques will now  
6 be described with reference to certain examples of operation. Of course, these  
7 examples are for illustrative purposes only and are indeed not exhaustive of the  
8 various applications of these mechanisms and techniques.

9  
10 Fig. 5 is a graphical illustration of a synchronization between a master  
11 namespace (DSA 150) and a slave namespace (buffer 221) that suffers from name  
12 collision. Shown are a data source (i.e., DSA 150) acting as a master namespace,  
13 and a buffer 221 acting as a slave. The sequence of events occurs from top to  
14 bottom, where the state of the DSA 150 changes over time. Initially, at state 501,  
15 the DSA 150 includes a first entity 510 currently named Alpha, and a second  
16 entity 511 currently named Beta. At that point, the second entity 511 is renamed  
17 from Beta to Tango, resulting in state 502. Subsequently, the first entity 510 is  
18 renamed to Beta.

19  
20 At that point an incremental synchronization occurs between the DSA 150  
21 and the buffer 221. Initially, at state 521, the buffer 221 matches the initial  
22 state 501 of the DSA 150. However, the changes to entities within the DSA 150  
23 are transmitted to the buffer 221 out of order. Thus, the first change that occurs in  
24 the buffer 221 is that the first entity 530 is renamed from Alpha to Beta, thus  
25 resulting in a name conflict. Note that the second entity 531 has not yet been

1 renamed. Accordingly, the second entity 531 is identified as transient in an  
2 appropriate manner, resulting in state 522. As described above, the name of the  
3 second entity 531 is changed in some fashion that prevents the two entities from  
4 sharing the same name. Subsequently, a change is received that renames the  
5 second entity 531 from Beta to Tango. Thus, the transient state of the second  
6 entity 531 is changed to non-transient, and its name is changed to Tango, resulting  
7 in state 523.

8  
9 Note that when the synchronization is complete, the final state 523 of the  
10 buffer 221 matches the final state 503 of the DSA 150, and there are no remaining  
11 artifacts (i.e., there are no remaining transients).

12  
13 Fig. 6 is a graphical illustration of another synchronization between a  
14 master namespace (DSA 150) and a slave namespace (buffer 221) that also suffers  
15 from name collision. Shown again are a DSA 150 acting as a master namespace,  
16 and a buffer 221 acting as a slave. The sequence of events occurs from top to  
17 bottom, where the state of the DSA 150 changes over time. Initially, at state 601,  
18 the DSA 150 includes only a first entity 610 currently named Alpha. At that point,  
19 the first entity 610 is deleted, and the DSA 150 is empty at state 602.  
20 Subsequently, at state 603, the second entity 611 is created and named Alpha.

21  
22 At that point an incremental synchronization occurs between the DSA 150  
23 and the buffer 221. Initially, at state 621, the buffer 221 matches the initial state  
24 601 of the DSA 150. However, the changes to entities within the DSA 150 are  
25 transmitted to the buffer 221 out of order. Thus, the first change that occurs in the

1 buffer 221 is that the second entity 631 is created and named Alpha, thus resulting  
2 in a name collision with the first entity 630. Accordingly, the first entity 630 is  
3 identified as transient in an appropriate manner, resulting in state 622. As  
4 described above, the name of the first entity 630 is also changed in some fashion  
5 that prevents the two entities from sharing the same name. Subsequently, a  
6 change is received that deletes the first entity 630, resulting in state 623.

7  
8 Note that when the synchronization is complete, the final state 623 of the  
9 buffer 221 matches the final state 603 of the DSA 150, and there are no remaining  
10 artifacts (i.e., there are no remaining transients).

11  
12 Fig. 7 is a graphical illustration of a synchronization between a master  
13 namespace (DSA 150) and a slave namespace (buffer 221) that suffers from a  
14 dangling reference. Shown again are a DSA 150 acting as a master namespace,  
15 and a buffer 221 acting as a slave. The sequence of events occurs from top to  
16 bottom, where the state of the DSA 150 changes over time. Initially, at state 701,  
17 the DSA 150 includes a first entity 710 currently named Alpha. At that point, a  
18 second entity 711 named Beta is created, resulting in state 702. Subsequently, a  
19 reference is added to the first entity 710 that points to the second entity 711.

20  
21 At that point an incremental synchronization occurs between the DSA 150  
22 and the buffer 221. Initially, at state 721, the buffer 221 matches the initial state  
23 701 of the DSA 150. However, the changes to entities within the DSA 150 are  
24 transmitted to the buffer 221 out of order. Thus, the first change that occurs in the  
25 buffer 221 is that a reference is added to the first entity 730 that points to the

1 second entity 731, but the second entity 731 has not yet been created.  
2 Accordingly, a phantom entity 731 is created having the name referred to by the  
3 first entity 730. As described above, a flag or bit within a typical entity may be  
4 used to indicate that the phantom entity 731 is a placeholder. Subsequently, a  
5 change is received that formally creates the second entity 731, and thus the  
6 phantom status is removed from it, resulting in state 723.

7  
8 Note that when the synchronization is complete, the final state 723 of the  
9 buffer 221 matches the final state 703 of the DSA 150, and there are no remaining  
10 artifacts (i.e., there are no phantom entities).

11  
12 Under normal conditions, a successfully completed synchronization process  
13 should leave the slave namespace (e.g., the buffer 221) in such a state that it has  
14 no transients or phantoms. This follows directly from the definition of a  
15 synchronization process, which stipulates that after a successfully completed  
16 synchronization the master and slave namespace are identical.

17  
18 If the slave namespace is left in a state with transients or phantoms, an error  
19 conditions occurs. If a synchronization process has terminated abnormally, then  
20 the application could simply warn the user about the presence of these entities,  
21 since subsequent resumption of the synchronization process should resolve their  
22 presence.

23  
24 A different action may be appropriate if either transients or phantoms  
25 remain after a synchronization process has completed successfully. This situation

1 can be result from two scenarios. First, the presence of transients after a  
2 successful synchronization process means the synchronization feed contains  
3 invalid data, and an error should be raised. The presence of phantoms after a  
4 successful synchronization would also mean the synchronization feed is invalid, as  
5 long as the synchronization process is always synchronizing all entities in both the  
6 master and slave namespaces.

7  
8 However, phantoms can remain after a successful synchronization if the  
9 synchronization process is modified such that only a subset of entities in the  
10 master namespace is synchronized with a corresponding subset in the slave  
11 namespace. Such a process may be called filtered synchronization, and it differs  
12 from normal synchronization in that it is possible for entities within the filtered  
13 subset to contain references to entities outside the filtered subset. Otherwise,  
14 filtered synchronization is identical to normal synchronization, except that the  
15 constraints on the process with respect to convergence only apply to the  
16 synchronized subsets of the master and slave namespaces. When the techniques  
17 described here operate on a filtered synchronization process, it is possible that  
18 phantom entities will remain if there are such references. From this perspective,  
19 phantoms may in fact be considered a useful tool in the representation of  
20 references to filtered entities in the underlying data source.

21  
22 Fig. 8 shows an exemplary computer 800 suitable as an environment for  
23 practicing various aspects of subject matter disclosed herein. Components of  
24 computer 800 may include, but are not limited to, a processing unit 820, a system  
25 memory 830, and a system bus 821 that couples various system components

1 including the system memory 830 to the processing unit 820. The system bus 821  
2 may be any of several types of bus structures including a memory bus or memory  
3 controller, a peripheral bus, and a local bus using any of a variety of bus  
4 architectures. By way of example, and not limitation, such architectures include  
5 Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA)  
6 bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association  
7 (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known  
8 as the Mezzanine bus.

9  
10 Exemplary computer 800 typically includes a variety of computer-readable  
11 media. Computer-readable media can be any available media that can be accessed  
12 by computer 800 and includes both volatile and nonvolatile media, removable and  
13 non-removable media. By way of example, and not limitation, computer-readable  
14 media may comprise computer storage media and communication media.  
15 Computer storage media include volatile and nonvolatile, removable and non-  
16 removable media implemented in any method or technology for storage of  
17 information such as computer-readable instructions, data structures, program  
18 modules, or other data. Computer storage media includes, but is not limited to,  
19 RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM,  
20 digital versatile disks (DVD) or other optical disk storage, magnetic cassettes,  
21 magnetic tape, magnetic disk storage or other magnetic storage devices, or any  
22 other medium which can be used to store the desired information and which can be  
23 accessed by computer 800. Communication media typically embodies computer-  
24 readable instructions, data structures, program modules or other data in a  
25 modulated data signal such as a carrier wave or other transport mechanism and

1 includes any information delivery media. The term "modulated data signal"  
2 means a signal that has one or more of its characteristics set or changed in such a  
3 manner as to encode information in the signal. By way of example, and not  
4 limitation, communication media includes wired media such as a wired network or  
5 direct-wired connection and wireless media such as acoustic, RF, infrared and  
6 other wireless media. Combinations of any of the above should also be included  
7 within the scope of computer readable media.

8  
9 The system memory 830 includes computer storage media in the form of  
10 volatile and/or nonvolatile memory such as read only memory (ROM) 831 and  
11 random access memory (RAM) 832. A basic input/output system 833 (BIOS),  
12 containing the basic routines that help to transfer information between elements  
13 within computer 800, such as during start-up, is typically stored in ROM 831.  
14 RAM 832 typically contains data and/or program modules that are immediately  
15 accessible to and/or presently being operated on by processing unit 820. By way  
16 of example, and not limitation, Fig. 8 illustrates operating system 834, the  
17 exemplary rules/specifications, services, storage 801 (e.g., storage may occur in  
18 RAM or other memory), application programs 835, other program modules 836,  
19 and program data 837. Although the exemplary rules/specifications, services  
20 and/or storage 801 are depicted as software in random access memory 832, other  
21 implementations may include hardware or combinations of software and hardware.

22  
23 The exemplary computer 800 may also include other removable/non-  
24 removable, volatile/nonvolatile computer storage media. By way of example only,  
25 Fig. 8 illustrates a hard disk drive 841 that reads from or writes to non-removable,

1 nonvolatile magnetic media, a magnetic disk drive 851 that reads from or writes to  
2 a removable, nonvolatile magnetic disk 852, and an optical disk drive 855 that  
3 reads from or writes to a removable, nonvolatile optical disk 856 such as a CD  
4 ROM or other optical media. Other removable/non-removable, volatile/nonvolatile  
5 computer storage media that can be used in the exemplary operating environment  
6 include, but are not limited to, magnetic tape cassettes, flash memory cards, digital  
7 versatile disks, digital video tape, solid state RAM, solid state ROM, and the like.  
8 The hard disk drive 841 is typically connected to the system bus 821 through a  
9 non-removable memory interface such as interface 840, and magnetic disk drive  
10 851 and optical disk drive 855 are typically connected to the system bus 821 by a  
11 removable memory interface such as interface 850.

12  
13 The drives and their associated computer storage media discussed above  
14 and illustrated in Fig. 8 provide storage of computer-readable instructions, data  
15 structures, program modules, and other data for computer 800. In Fig. 8, for  
16 example, hard disk drive 841 is illustrated as storing operating system 844,  
17 application programs 845, other program modules 846, and program data 847.  
18 Note that these components can either be the same as or different from operating  
19 system 834, application programs 835, other program modules 836, and program  
20 data 837. Operating system 844, application programs 845, other program  
21 modules 846, and program data 847 are given different numbers here to illustrate  
22 that, at a minimum, they are different copies. A user may enter commands and  
23 information into the exemplary computer 800 through input devices such as a  
24 keyboard 862 and pointing device 861, commonly referred to as a mouse,  
25 trackball, or touch pad. Other input devices (not shown) may include a



1 microphone, joystick, game pad, satellite dish, scanner, or the like. These and  
2 other input devices are often connected to the processing unit 820 through a user  
3 input interface 860 that is coupled to the system bus, but may be connected by  
4 other interface and bus structures, such as a parallel port, game port, or a universal  
5 serial bus (USB). A monitor 891 or other type of display device is also connected  
6 to the system bus 821 via an interface, such as a video interface 890. In addition  
7 to the monitor 891, computers may also include other peripheral output devices  
8 such as speakers 897 and printer 896, which may be connected through an output  
9 peripheral interface 895.

10  
11 The exemplary computer 800 may operate in a networked environment  
12 using logical connections to one or more remote computers, such as a remote  
13 computer 880. The remote computer 880 may be a personal computer, a server, a  
14 router, a network PC, a peer device or other common network node, and typically  
15 includes many or all of the elements described above relative to computer 800,  
16 although only a memory storage device 881 has been illustrated in Fig. 8. The  
17 logical connections depicted in Fig. 8 include a local area network (LAN) 871 and  
18 a wide area network (WAN) 873, but may also include other networks. Such  
19 networking environments are commonplace in offices, enterprise-wide computer  
20 networks, intranets, and the Internet.

21  
22 When used in a LAN networking environment, the exemplary computer  
23 800 is connected to the LAN 871 through a network interface or adapter 870.  
24 When used in a WAN networking environment, the exemplary computer 800  
25 typically includes a modem 872 or other means for establishing communications

1 over the WAN 873, such as the Internet. The modem 872, which may be internal  
2 or external, may be connected to the system bus 821 via the user input interface  
3 860, or other appropriate mechanism. In a networked environment, program  
4 modules depicted relative to the exemplary computer 800, or portions thereof, may  
5 be stored in the remote memory storage device. By way of example, and not  
6 limitation, Fig. 8 illustrates remote application programs 885 as residing on  
7 memory device 881. It will be appreciated that the network connections shown  
8 are exemplary and other means of establishing a communications link between the  
9 computers may be used.

10  
11 The subject matter described above can be implemented in hardware, in  
12 software, or in both hardware and software. In certain implementations, the  
13 exemplary flexible rules, identity information management processes, engines, and  
14 related methods may be described in the general context of computer-executable  
15 instructions, such as program modules, being executed by a computer. Generally,  
16 program modules include routines, programs, objects, components, data structures,  
17 etc. that perform particular tasks or implement particular abstract data types. The  
18 subject matter can also be practiced in distributed communications environments  
19 where tasks are performed over wireless communication by remote processing  
20 devices that are linked through a communications network. In a wireless network,  
21 program modules may be located in both local and remote communications device  
22 storage media including memory storage devices.